

# Feasibility Study Book Reader

15 September 2009

Commissioned by John Reyer Afamasaga  
Written by Simon Judge, Freelance Mobile Developer

# Contents

- 1 Overview ..... 3
- 2 Introduction ..... 3
  - 2.1 Possible Types of Implementation ..... 3
  - 2.2 The Book Reader Web Server ..... 3
  - 2.3 Background Running ..... 4
  - 2.4 Notification via SMS ..... 4
  - 2.5 Digital Signing ..... 4
  - 2.6 Video Formats ..... 4
  - 2.7 Audio Formats ..... 4
- 3 Symbian OS ..... 5
  - 3.1 Versions ..... 5
  - 3.2 Application Signing ..... 5
  - 3.3 Functionality ..... 5
- 4 iPhone ..... 5
- 5 Android ..... 6
- 6 Windows Mobile ..... 6
  - 6.1 Versions ..... 6
  - 6.2 Signing ..... 6
  - 6.3 Functionality ..... 6
- 7 Java ME ..... 7
  - 7.1 Versions ..... 7
  - 7.2 Application Signing ..... 7
  - 7.3 Functionality ..... 8
- 8 BlackBerry ..... 9
  - 8.1 Versions ..... 9
  - 8.2 Functionality ..... 9
- 9 Application Distribution ..... 9
  - 9.1 Wap Push ..... 9
  - 9.2 Side Loading ..... 9
- 10 The Market ..... 10
  - 10.1 Geography ..... 10
- 11 Testing ..... 10
- 12 Time Estimates ..... 11
- 13 Conclusions ..... 11

# 1 Overview

This document is a feasibility study into a multi-platform book reader application.

These are the main requirements considered in this document:

- Advertisements within the book text
- Delivery of installments without re-install of the application
- Alerting when a new installment is available
- Installment text of the order of thousands of characters
- Accessibility to old installments
- Playing of media types other than text (audio, video, flash, images)
- Application Distribution
- The market for applications
- Application testing

The following platforms are considered:

- Symbian OS
- iPhone
- BlackBerry
- Windows Mobile
- Java ME
- Android

## 2 Introduction

### 2.1 Possible Types of Implementation

There are three main ways the application could be implemented:

- *Rich* application. This would do everything itself, particularly the display of media types other than text.
- *Delegation* application. This would delegate the display of media, other than text, to the phone web browser or other built-in application associated with that file type.
- *Shell* application. This would not contain the text itself but could be thought of as a very simple web browser within the application. It would render information fetched from a remote book reader web server. It would require a phone data connection at the point of reading an installment. It would have the advantage that book/advertising data could easily be changed at the server and older installments would automatically be available via links. Again, the application would delegate the display of complex media other than text to the more capable phone web browser.

### 2.2 The Book Reader Web Server

All the above solutions would require a book reader web server in order to host book and advert data. It's envisaged this would be simple, inexpensive shared web hosting or even part of an existing web site.

## 2.3 Background Running

In order to alert users of new installments it's necessary for the application to always be running in the background. Furthermore, it needs to start when the phone starts ('start at boot').

It's envisaged a background book reader application would check periodically (every day?) with the book reader server for new installments. If a new installment were found it could be optionally automatically downloaded and alerted to the user.

## 2.4 Notification via SMS

Where background running isn't available, alerts might be sent from the book reader web server via SMS. This would require that the user to register in some way for SMS alerts from the book web server. This might occur via extra screens within the phone application, possibly when they first run the application. Sending SMS messages from the server would complicate the book reader web server design/implementation and also incur extra ongoing running cost.

If SMS messages were sent from the book reader web server then this would most likely occur via a SMS aggregator such as Clickatell ([www.clickatell.com](http://www.clickatell.com)).

## 2.5 Digital Signing

Operating systems often require that applications be digitally signed so as to prove their origin. This is discussed later in this document insofar as it affects development of the applications.

## 2.6 Video Formats

Phones cannot display full flash content as presented by normal PC/Mac web pages. Instead, they use a different video formats depending on the actual phone model. Some phone support Flash Lite, a specially authored variant of Flash for mobile. To complicate matters, this also comes in three variants, Flash Lite 1.1, 2 and 3. Phones sometimes also support .3gp and/or .mp4. These aren't actually video formats but ways in which the video is packaged. Think of them as an outside wrapper. The important part is inside the wrapper that is usually H263, H264 or MPEG-4 encoding that phone platforms may or may not support. Furthermore, video is further complicated in that video authored for the PC/Mac isn't usually suitable for mobile, even if the format can be read directly, because of differing screen size. It's expensive and time consuming for users to download the large files authored for much larger screens.

In order to use video within a *rich*, *delegation* or *shell* application the video would have to be re-authored on a page on the book reader web site and linked to from the book reader application. There would also have to be different types of video depending on the types of phone to be supported.

The book reader application would send a 'hint' (via the http header) to the book reader web server that would allow it to determine the phone type. The server would then return the appropriate video type. There are standard phone capability databases (e.g. WURFL <http://wurfl.sourceforge.net/> and .mobi DeviceAtlas <http://deviceatlas.com/>) that might be used to identify phones and their capabilities at the web server.

## 2.7 Audio Formats

All the proposed platforms support .mp3 so this isn't anticipated to be a problem. The only problem might be with older Java ME phones that may not support the .mp3 format.

## 3 Symbian OS

### 3.1 Versions

There are two main versions of Symbian S60 in circulation, the older S60 2<sup>nd</sup> and newer (since 2005) S60 3<sup>rd</sup>. Applications for each are not compatible and need to be developed separately although some code can be shared.

There's also a UIQ variant of Symbian, championed by Sony Ericsson. However, since Nokia purchased Symbian and created the Symbian Foundation to open source the Symbian OS, UIQ Technology (the company) has closed and phones will no longer be produced using the UIQ Symbian variant. Relatively very few phones were shipped using the UIQ variant.

I'd recommend that the book reader only be developed for S60 3<sup>rd</sup> unless targeting developing countries where S60 2<sup>nd</sup> is still ubiquitous.

### 3.2 Application Signing

S60 3<sup>rd</sup> introduced the need to sign applications. Either self-signed (which costs and implies nothing) or Symbian Signed which involves additional testing against pre-defined criteria. Symbian Signed allows the application to access protected facilities. S60 2<sup>nd</sup> applications do not need to be signed.

For the S60 3<sup>rd</sup> (but not S60 2<sup>nd</sup>), start at boot is only possible if the application has been Symbian Signed. This is to ensure rogue applications don't permanently disable a phone.

### 3.3 Functionality

Both S60 2<sup>nd</sup> and S60 3<sup>rd</sup> support all the requirements. They could be used to develop a *rich*, *delegation* or *shell* application. It's possible to literally embed a web viewer in an application so a *shell* application would be particularly simple to implement.

Having said this, Symbian development itself is the most complex type of development. Additionally, there's extra code, testing and red tape involved in Symbian Signing. A *rich* application would be relatively expensive to produce.

Symbian devices have ample memory and a whole book (in fact many books) could be stored on the phone.

Symbian phones support Flash Lite, H263 and MPEG 4 VSP.

## 4 iPhone

The iPhone could be used to develop a *rich*, *delegation* or *shell* application. As with S60, it's possible to embed a web viewer in an application so a *shell* application would be particularly simple to implement.

The iPhone doesn't support start at boot and background applications. 3<sup>rd</sup> party applications are run one at a time. The currently running application is stopped whenever a new one is started. Apple has a notification system where a server (the book reader server) can send an update to the Apple server

that sends the notification on to the phone. Alternatively, SMS messages could be used as previously mentioned.

iPhone devices have ample memory and a whole book (in fact many books) could be stored on the phone.

For video, the iPhone supports MPEG-4 SP and H264 within a .3gp wrapper.

## 5 Android

Android could be used to develop a *rich*, *delegation* or *shell* application. As with S60 and iPhone, it's possible to embed a web viewer in an application so a *shell* application would be particularly simple to implement.

Android supports auto-start and works as a background application if it is implemented as an Android service. Services can't have a UI so the application would also need a separate application (called an Activity) to deal with user input. It's possible to start a service at phone boot time.

iPhone devices have ample memory and a whole book (in fact many books) could be stored on the phone.

For video, Android supports MPEG-4 SP and H264/H263 within a .3gp or .mp4 wrapper.

## 6 Windows Mobile

### 6.1 Versions

Windows Mobile provides two phone platforms. The Pocket PC variant (also called Windows Mobile Professional) is PDA centric and has a touch screen. The Smartphone variant (also Pocket PC Standard) does not have a touchscreen and the UI is designed for single hand use.

Both platforms share common programming interfaces, part of which is protected and only available to applications signed and 3<sup>rd</sup> party tested via the Mobile2Market scheme.

### 6.2 Signing

The programming interfaces required by the application would not require signing. However, smartphone devices supplied (and sometimes branded) by carriers, as opposed to purchased without a tariff, tend to be 'application locked' in that it's not possible to install unsigned applications. It's down to the particular carrier to determine whether to application lock a device. Some also provide end-user unlock tools. It's sometimes possible for the user to circumvent the application locking but this isn't something that the typical user could be expected to do.

Hence, to reach the maximum market, particularly that of devices provided by carriers, a Windows Mobile Application might be signed via Mobile2Market.

### 6.3 Functionality

Both the Pocket PC and Smartphone variants support all the requirements. They could be used to develop a *rich*, *delegation* or *shell* application. It's possible to embed a web viewer in an application so a *shell* application would be particularly simple to implement.

Windows Mobile applications can be auto-started and remain in background. They also have ample memory and a whole book (in fact many books) could be stored on the phone.

Video support on Windows Mobile depends on what the end phone manufacturer has included. Although the common mobile video formats aren't included in Windows Mobile by default, many manufacturers add them for their particular phones.

## **7 Java ME**

### **7.1 Versions**

There are three main versions of Java that run on phones...

- MIDP 1.0 was a very early version with less programming features.
- MIDP 2.0 represents the bulk of current Motorola, Sony Ericsson and Nokia S40 phones.
- MIDP 2.1 is found on only the very latest phones.

It's recommended that development target MIDP 2.0 as this is forward compatible with MIDP 2.1.

### **7.2 Application Signing**

Java Signing is complex and has many facets. Here's a summary.

There are restrictions for accessing some APIs from Java applications. In these cases the user will either be prompted for confirmation to allow a certain method call or the access is blocked altogether, resulting in an error.

Making these prompts appear less frequently requires the developer to sign the application. Applications can be one of...

- Unsigned
- Signed 3<sup>rd</sup> Party (usually Java Verified)
- Operator (carrier) certificate signed
- Manufacturer (handset OEM) certificate signed

These classifications are called 'domains'.

When accessing protected functionality such as accessing the Internet or files on the phone, the behavior depends on the signing domain for the application. The application may either...

- not allow access
- ask the user every time
- ask only the first time or
- always allow

...depending on the functionality being accessed.

There must be a corresponding (Java Verified , Operator, Manufacturer) certificate on the phone in order for the signed application to be recognized a such.

3<sup>rd</sup> party (typically Java Verified) signing requires that the application be submitted and pass a series of tests.

While **operator** certificates are widely available on most devices, they are not available on all devices. There are many regional and operator-specific differences due to parties omitting or adding certificates or permissions at time of phone manufacture. Some examples documented by Nokia include...

- AT&T Java security domains (Cingular)
- China Unicom Java security domains
- Hutchinson 3G security domains - note, that Orange Israel follows the Hutchinson 3G guidelines too
- Sprint Java security domains
- T-Mobile U.S. Java security domains

**Operator** signing requires close partnering with that particular operator and rarely happens in practice. **Manufacturer** signing requires the application be branded as a manufacturer (eg Nokia) or that the application be developed by the manufacturer.

The summary of all of this is that there will always be cases where Java ME applications will present annoying prompts to the user to access files or access the Internet.

### 7.3 Functionality

Java provides the most challenges because it is the least capable platform and also tends to run on less powerful phones.

Ordinarily, it's not possible to auto-start Java ME applications at boot time unless, for example with Motorola and some Sony Ericsson phones, the phone OEM has provided special facilities to do this. Auto start should eventually be part of MIDP 3.0 that isn't found in any phones yet.

Also, it's not possible to guarantee that a Java application will remain running. On most phones only one Java ME application can run at any one time. Even on more powerful smartphones, where multiple Java ME applications can run concurrently, these may be closed down when the phone is short of memory.

Java suffers from fragmentation of functionality across different Java phones. What may work on one phone model may not work on others. Hence, while producing a *rich* application for one phone model might be easy, testing and adapting for other phone models would be prohibitively expensive. A *delegation* application would be more achievable as it would use less of Java's capabilities. A *shell* application would be difficult because there's no way to embed a web page within a Java application.

In addition to functionality fragmentation, Java also suffers from variance of memory and screen size across phones. Hence, it's not possible to guarantee that a whole book might fit into the memory available to Java. Additional screens would be required to choose and fetch old installments. More recently, a Java library called LWUIT has become available that abstracts away the differences in screen size while also providing better look and feel. This is mentioned further in the conclusions.

Some Java phones support Flash Lite and most recent MIDP 2.0 devices support .3gp and MPEG-4.

## 8 BlackBerry

### 8.1 Versions

BlackBerry has many versions and most current applications use the v4.2 SDK that allows support for all recent phones while not going back so far as to produce problems with fragmentation of functionality.

BlackBerry is based on Java ME. It's a superset of Java ME and runs on more powerful phones. This means that many of the problems associated with Java ME do not exist on BlackBerry. For example, signing is very simple and just involves purchase of an inexpensive signing key.

### 8.2 Functionality

BlackBerry supports all the requirements. It could be used to develop a *rich*, *delegation* or *shell* application.

It's possible to auto-start an application and have it sit in the background waiting for phone events. Hence, this can be used to periodically check for new installments.

BlackBerry devices have ample memory and a whole book (in fact many books) could be stored on the phone. BlackBerry phones support 3GP and MPEG 4.

## 9 Application Distribution

Up until very recently, applications were either distributed either via owners' own web sites, via Handango (<http://www.handango.com>) or via carriers' own portals. Today, distribution has moved to one of the many application stores:

Symbian – Ovi.  
Java ME – getjar.com  
Android – Android Market  
iPhone – App Store  
BlackBerry – BlackBerry App World  
Windows Mobile – Windows Mobile Market

### 9.1 Wap Push

With the exception iPhone, it's also possible to supply applications via the owner's web site or via a SMS message.

Applications are usually sent to a phone via a mechanism called 'WAP push'. This is a specially coded SMS message with a link to download the application. The phone recognizes the special message and asks the user if they wish to download the application. The same mechanism used to download a WAP page is then used to download the application. Unfortunately, WAP push isn't reliable across all phone types. Some phones need slightly different special messages, others have bugs and some don't even recognize these messages. Hence, content providers tend to also send a normal SMS containing the link. Users can download by clicking on the link invoking a similar process as per WAP push.

### 9.2 Side Loading

A more reliable method of getting applications onto phones is via cable, Infra Red or Bluetooth from a

connected PC. Most Symbian and Windows Mobile owners expect to be able to download this way.

## 10 The Market

The latest Q209 worldwide statistics from Canals suggest market shares as follows...

Symbian 50.3%  
BlackBerry 20.9%  
Apple 13.7%  
Microsoft 3.4%  
Others 1.2%

This represents about 150 million devices per year. To put this into perspective, of the order of 1 billion phones are sold every year, a large number of which (exact statistics unavailable) are Java phones. However, unfortunately a large proportion of Java phones sold don't have data tariffs that allow 3<sup>rd</sup> party applications to be downloaded.

### 10.1 Geography

There are also some geographical factors that might affect choice of platform. If predominantly selling to North America, iPhone, Windows Mobile and BlackBerry should be developed. Very few S60 devices are sold in North America. If predominately selling to developing countries, S60 2<sup>nd</sup> device should be considered.

## 11 Testing

There are generally four approaches to testing (for production rather than proof of concept)...

1. Get users to beta test. This can be frustrating for users but can provide an inexpensive way of supporting a large number of devices quickly.
2. Use a third party test house. This typically costs several hundreds of pounds per device per test. This can quickly become very expensive as the number of devices are scaled up and if there are many iterations of the software.
3. Test in-house, purchasing devices for test. This is the preferred solution if costs allow because it also allows (user as opposed to application) problems to be more easily diagnosed when the application is in service.
4. A solution between #2 and #4 is use of a facility such as Paca Mobile Center <http://www.pacamobilecenter.com/> that provides access to test phones on a daily or annual basis.

## 12 Time Estimates

The following time estimates for design/development are man days and assume only functionality as described in this document. Time also excludes any book reader web site work.

	Rich App	Delegate App	Shell App
<b>Symbian</b>	55	23	15
<b>Android</b>	44	15	10
<b>Windows Mobile</b>	44	15	10
<b>Java ME</b>	> 6 months*	15	30
<b>BlackBerry</b>	44	15	10

\* Java version assumes support for the majority of Java phones currently on the market

## 13 Conclusions

A *rich* application would be very costly to develop, especially given that the application is required to be developed for many platforms. Furthermore, the ongoing cost of supporting and developing for new variants of platforms would also be large (platforms are evolving fast and current applications soon become out of date).

It's recommended that the application be as simple as possible while maintaining the core requirement to not have to go online at the point the user wants to read content. Unlike many other mobile applications, people would be downloading for the content (i.e. the book) not the application itself. Hence, the application is not competing with others on a per-feature basis. Hence, it's recommended that the *delegation* rather than *rich* application approach be used.

Auto update via SMS would complicate the server side and may not be justified on non-supported platforms. It may be easier just to ask the user to do a manual update.

As per the *delegate* approach, it's recommended that media be presented as just text with a link to play the media in the phone web browser or other application associated with that file type. It's recommended that for simplicity, adverts also be treated the same way as links to media.

For complex media (i.e. Video) it will be necessary to pre re-author it to be compatible with specific phone models. The server will also have to detect the phone type from hints sent from the phone application and present the correct media type.

When implementing applications, it possible to use the controls (edit boxes, static text, tick boxes etc) provided by the platform or draw your own on a pixel-like screen. The latter has the advantage that much graphically richer applications, with better look and feel, can be produced with the disadvantage of considerably more development time. The time estimates in the document assume using controls provided by each respective platform. This is recommended because such controls automatically scale to different screen sizes and require much less work. The exception to this is Java ME where it's recommended that an extra free Sun library (LWUIT) be used to allow for graphically rich screens while retaining screen size independence.